Comparison of Python, Java, and C Performance in Matrix Multiplication

Álvaro Rodríguez González
Data Science and Engineering Student - ULPGC

20/10/24

Abstract

The objective of this task is to compare the performance of various programs written in different programming languages (Python, Java, and C) when executing a computationally expensive task, matrix multiplication. The key performance metrics for evaluating these programs are execution time and memory usage. The experimentation involved running the same algorithm in all three languages, measuring their performance across different matrix sizes (10, 100, and 1000). Results showed notable variations depending on the programming language, with some languages excelling in terms of execution speed but showing higher memory consumption, and others performing better in memory efficiency while taking longer to execute. These results emphasize the trade-offs inherent in language selection for computationally intensive tasks.

1 Introduction / Background / Motivation

Matrix multiplication is a fundamental operation in many computational tasks, particularly in fields like scientific computing, machine learning, and computer graphics. However, its performance can vary significantly depending on the programming language and algorithm implementation. Benchmarking is a common technique used to evaluate the efficiency of different programming languages when executing computationally expensive tasks like this one. Several studies have already explored how languages such as

Python, Java, and C handle computational operations, often focusing on specific metrics like execution time and memory usage.

This paper focuses on running a series of benchmarks to directly compare Python, Java, and C in matrix multiplication tasks. By measuring execution time and memory usage for varying matrix sizes, this study provides insights into the trade-offs between these languages, offering practical guidance for selecting the right language for computationally demanding tasks.

2 Problem Statement

Matrix multiplication is a computationally expensive task, especially as the size of the matrices increases. In this paper, I aim to compare how Python, Java, and C handle this task by benchmarking their performance with different matrix sizes: 10x10, 100x100, and 1000x1000.

3 Methodology

The matrix multiplication problem was approached by measuring how results differ depending on the matrix sizes. In both Java and Python, external libraries were used to track memory usage in kilobytes (KB), as this metric is not directly available in the default benchmarking results. In C, I relied not only on the performance outputs provided by the perf command but also explored where memory usage information is logged, specifically in a file located under /usr/bin/time.

The procedure is straightforward. For Java and Python, I configured the benchmarks by specifying the matrix sizes and the number of iterations to be tested. This provided a summary of the desired operations. In C, the process was slightly different; matrix sizes and iteration counts were manually set in a loop. In essence, I tried to mimic the behavior of the benchmarking systems in Python and Java for C by manually managing matrix sizes and iterations.

4 Experiments and Results

The results of our experiments are presented below, with time measured in milliseconds and memory usage in bytes (Java) or megabytes (Python) and kilobytes (C).

4.1 Java Results

The following table shows the results of matrix multiplication in Java, with execution time measured in milliseconds and memory usage in bytes.

Matrix Size	**Execution Time (ms)**	**Memory Usage (bytes)**
10	27.29	9638.83
100	82.73	111479.43
1000	25669.92	7944847.47

Table 1: Java Benchmark Results for Matrix Multiplication

As shown in Table 1, the execution time increases significantly as the matrix size grows, particularly for the largest matrix size of 1000x1000, where the execution time reaches 25.6 seconds. Memory usage also scales with matrix size, reaching nearly 8 MB for the largest matrix. This indicates that Java's memory usage increases linearly with matrix size, though the execution time grows more dramatically.

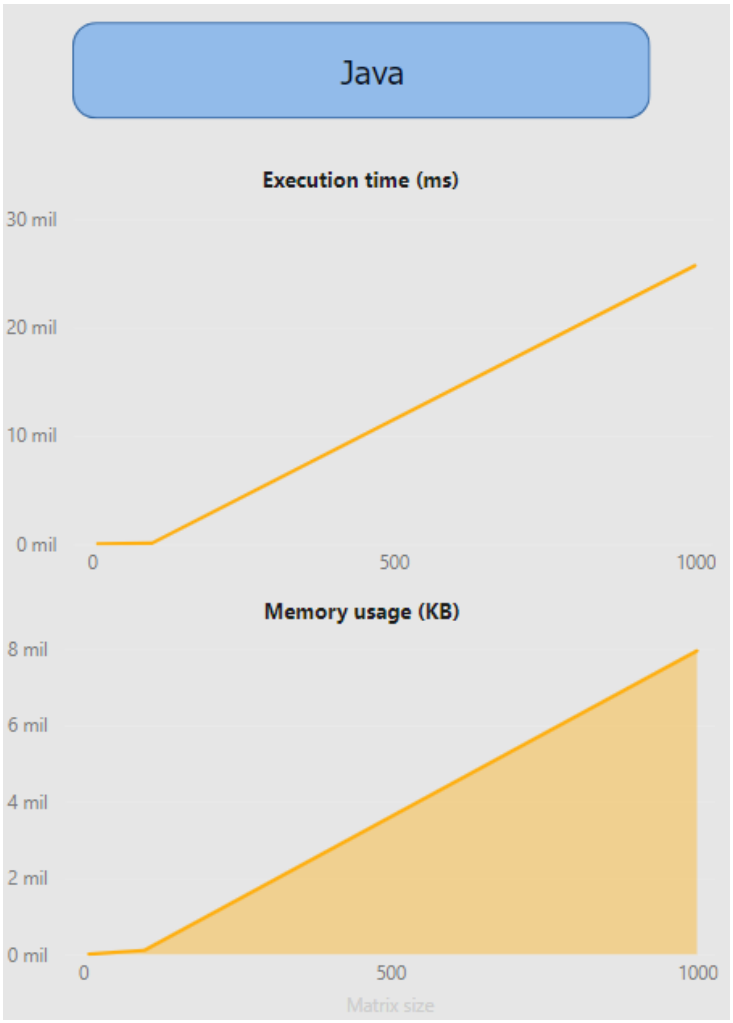

Figure 1: Graphical Representation of Java Results

4.2 Python Results

The following table shows the results of matrix multiplication in Python, with execution time measured in microseconds and memory usage in megabytes.

Matrix Size	Execution Time (microseg)	Memory Usage (MB)
10	61.74	68.2075893
100	4801.72	68.5783761
1000	477098.31	89.8828125

Table 2: Python Benchmark Results for Matrix Multiplication

In Table 2, we can see that Python's execution time grows exponentially with matrix size, especially for the largest matrix, where the execution time exceeds 477 milliseconds. Memory usage also increases with matrix size, although at a faster pace compared to Java. This suggests that Python, while easy to implement, suffers from higher memory increase in larger computations.

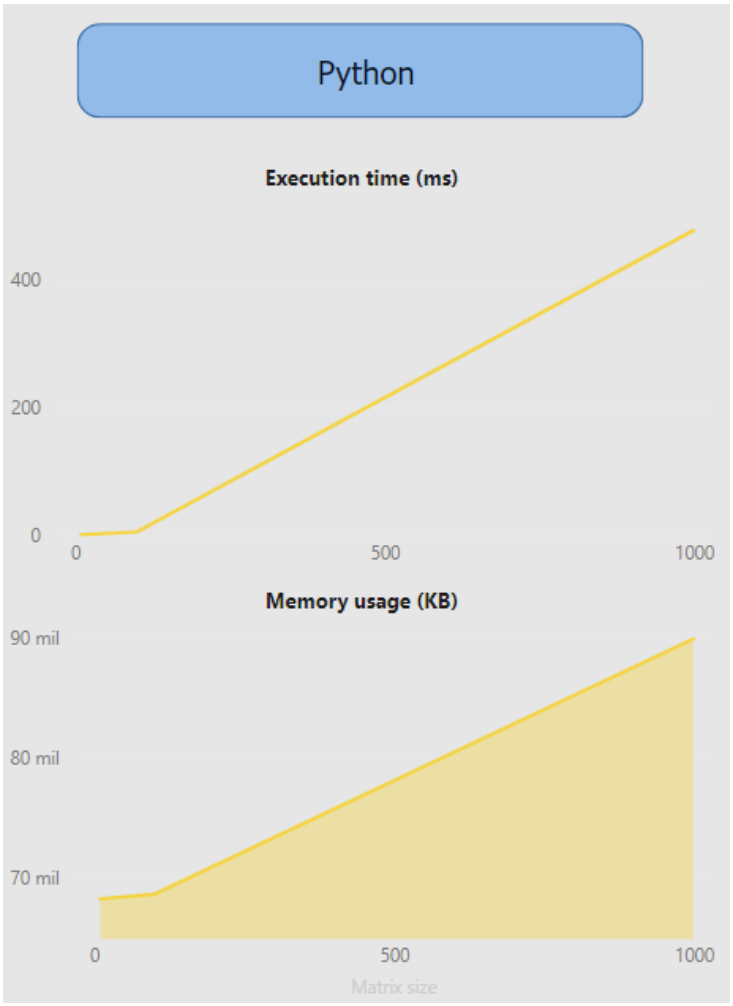

Figure 2: Graphical Representation of Python Results

4.3 C Results

The following table shows the results of matrix multiplication in C, with execution time measured in milliseconds and memory usage in kilobytes.

Matrix Size	Execution Time (ms)	Memory Usage (KB)
10	1193.80	525.54
100	1152.99	1466.51
1000	1156.30	1132.99

Table 3: C Benchmark Results for Matrix Multiplication (Averaged)

Table 3 illustrates that C maintains a relatively constant execution time for all matrix sizes, with only minor variations between 10x10 and 1000x1000 matrices. Memory usage also remains stable and low compared to Python and Java, making C the most efficient language in this experiment in terms of both execution time and memory usage.

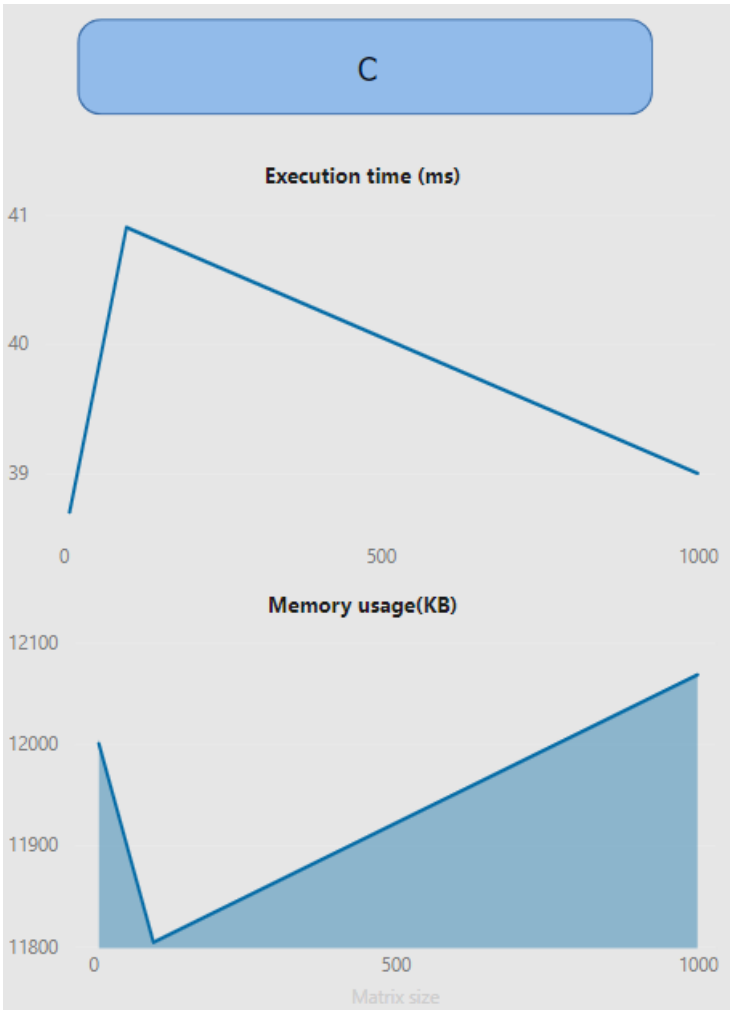

Figure 3: Graphical Representation of C Results

The figures above provides a visual comparison of execution times and memory usage across all three languages. Java shows the largest increase in both execution time with larger matrix sizes, while Python shows rapid growth in memory increase. C remains stable in both aspects, confirming its efficiency.

5 Conclusion

In this paper, we tackled the problem of comparing the performance of three different programming languages—Python, Java, and C—when performing matrix multiplication, a computationally expensive task. Our goal was to understand how each language handles this task and to measure two critical performance metrics: execution time and memory usage. By conducting benchmarks with varying matrix sizes (10x10, 100x100, and 1000x1000), we were able to obtain valuable insights into the strengths and weaknesses of each language in terms of computational efficiency.

The results of our experiments reveal several key findings. Python showed a steady increase in execution time as matrix size grew, with the largest matrix taking over 400 milliseconds, confirming Python's tendency to be slower in computationally intensive tasks. Memory usage in Python also increased, particularly for the largest matrix size, reaching nearly 90,000 KB. C, in contrast, demonstrated very efficient execution times, with a notable drop in time for larger matrix sizes. Interestingly, C maintained low and stable memory usage across different matrix sizes, never exceeding 12,000 KB, making it the most efficient in terms of memory management. Java exhibited a sharp increase in execution time, reaching up to 30,000 ms for the largest matrix, and its memory usage also escalated significantly, nearing 120,000 KB. These findings highlight the clear advantages of using C for both time and memory efficiency, while Java and Python show trade-offs, with Java being slower and Python consuming more memory as matrix sizes increase.

The importance of this experimentation lies in its practical implications. For developers and researchers who work on computationally intensive tasks, understanding the trade-offs between execution time and memory usage is essential. This study highlights the clear advantages of using C for matrix multiplication, particularly for larger datasets where performance is critical. Meanwhile, Python and Java may be more suitable for tasks where ease of use or development speed is prioritized over raw computational efficiency.

In conclusion, the results of this paper provide clear guidance on selecting the appropriate programming language for matrix multiplication tasks, depending on the specific performance requirements.

6 Future Work

In this project, I had to calculate memory usage for Java and Python using specific external libraries, as the benchmarking tools I used do not provide memory usage data directly. In contrast, for C, I retrieved memory usage information from a dedicated performance file generated by the system, which I believe provides more reliable and accurate results.

For future work, we should consider implementing matrix multiplication optimizers or utilizing external libraries that are specifically designed for efficient matrix operations, such as NumPy in Python. This could lead to more accurate and optimized results in Python, potentially improving its performance.

Additionally, it would be beneficial to conduct these experiments on different hardware configurations to gather a broader set of reference points. By testing on various systems, we could better understand how these languages scale across different computational environments, providing deeper insights into performance variability.

7 Additional Resources

7.1 GitHub Repository

All the code and data used in this project are available in the following GitHub repository:

GitHub Repository: Matrix Multiplication Performance Comparison

7.2 Power BI Dashboard

The visual representation of the experimental results can be explored through the Power BI dashboard:

Power BI Dashboard: Matrix Multiplication Results